Rubik’s Cube Solving Robot

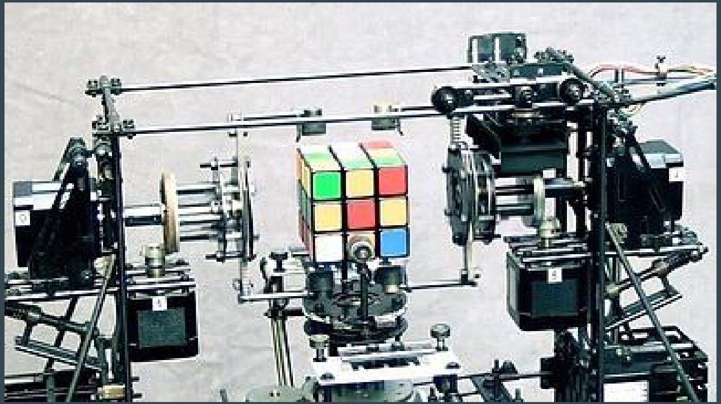
•••

Daniel Truesdell - EE
Corey Holsey - CpE
Tony Verbano - CpE
Group 12

Motivation

Build a robot that can solve a scrambled Rubik's Cube

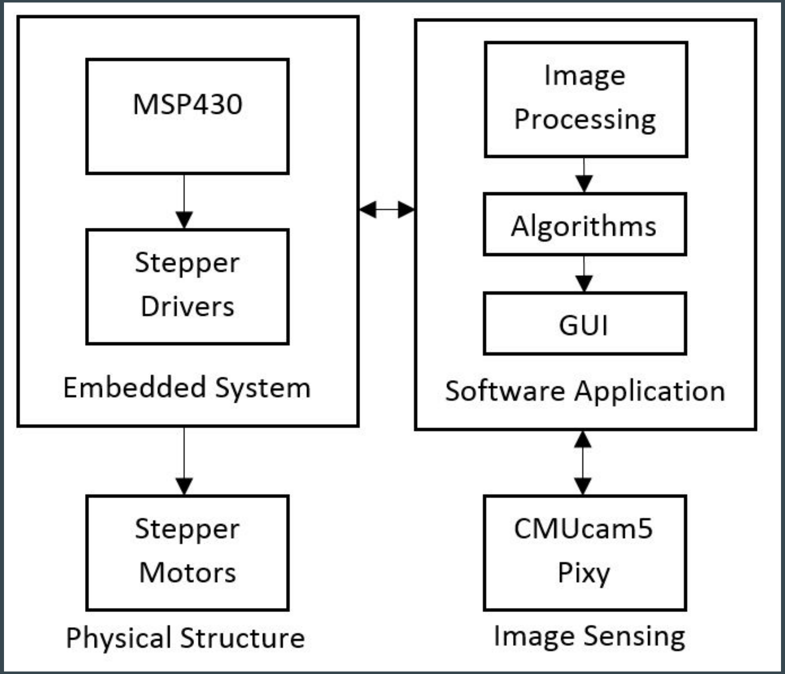
- Combination of hardware and software systems
- Rubik's Cube is a fascinating puzzle



blog.zok.pw

General

- Four separate parts of the project
  - Physical Structure
    - Stepper Motors
  - Embedded System
    - Motor control
    - Processor
  - Cube Visualization
    - CMUcam5 Pixy
    - Visualization implementation
    - Matrix Input
  - Rubik's cube algorithm
    - CFOP method Vs Kociemba
    - Mathematics
    - Randomization of the cube

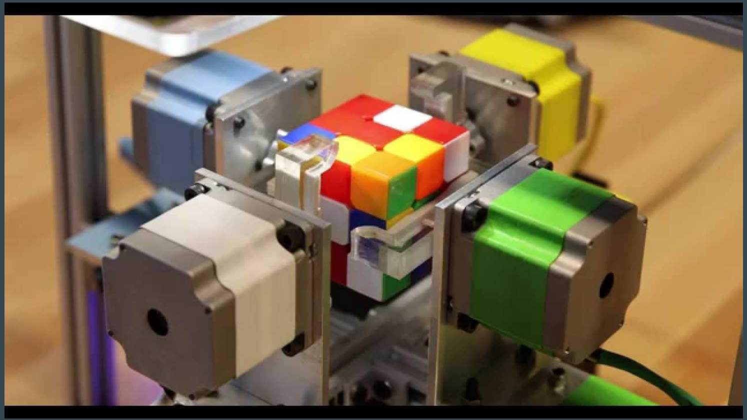


Desired Functions

- Solve a Rubik's cube correctly 90 % of the time
- Fully visualize and map the cube 90% of the time
- Solve the cube in at least 15 minutes
- Mechanical manipulation of cube in all ways
- Provide Graphical User Interface

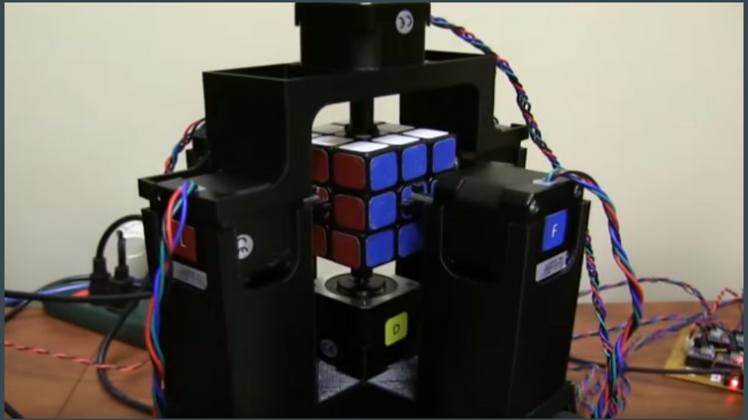
Mechanical Design - Structural Platform

Prototype



YouTube - Calit2ube (Raspberry Pi - based)

3D-Print



YouTube - Jay Flatland (PC-based)

Mechanical Design - Cube Control

- One motor per side of the cube
- No repositioning necessary
- No claw/gripper necessary
- Fastest cube manipulation

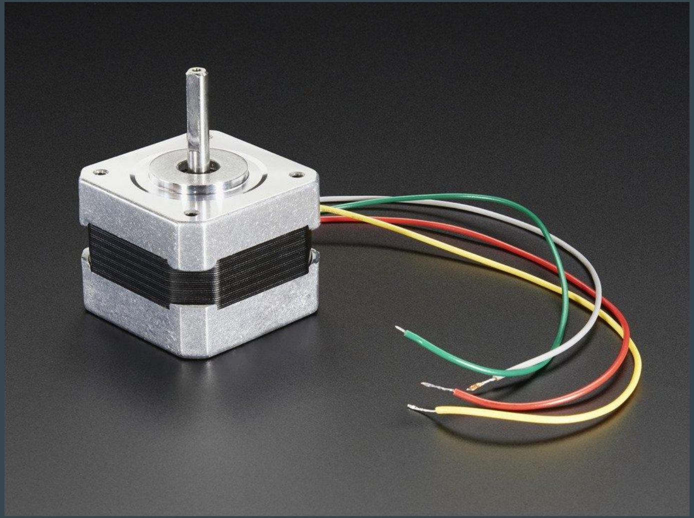
Mechanical Design - Motors

	DC	Servo	**Stepper**
Pros	High RPM Easy to operate	Easy to operate - Single PWM input	**Predetermined, reliable positioning**
Cons	Lack of position control	Require feedback or precise tuning for accurate positioning	**Usually require driver IC**

Mechanical Design - Motors

Adafruit NEMA-17 Stepper Motor

- $14 per motor
- 1.8° Step size = 200 steps per revolution
- Rated for 350mA at 12V
- Small, robust

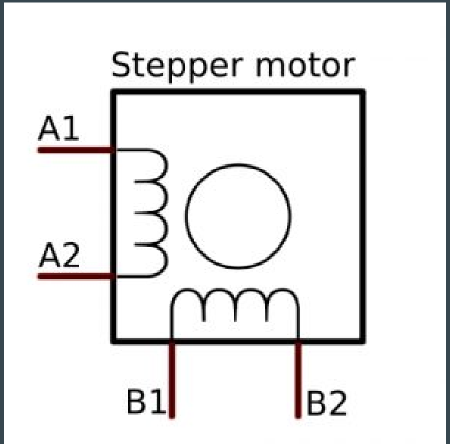


Adafruit.com

Electrical Design - Motor Control

Motor Control Requirements:

- Bidirectional motion requires bidirectional current (source/sink) ability on all 4 wires
- Smooth operation requires precise coil actuation and current control



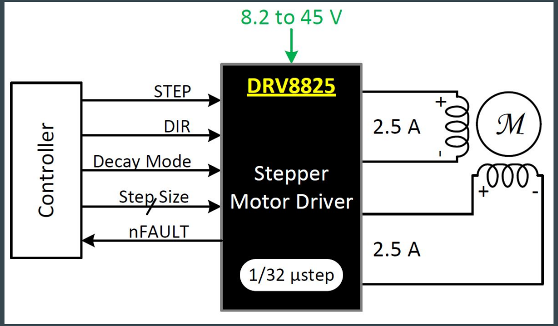
42bots.com

Electrical Design - Motor Control

Solution: TI DRV8825 Stepper Motor Driver

- 2.5A max current output
- Integrated H-bridge circuit for bidirectional motion
- Isolates processor from harmful back-EMF
- Allows separate (12V) motor supply voltage
- Simple control scheme (enable, step, direction)

Pololu.com

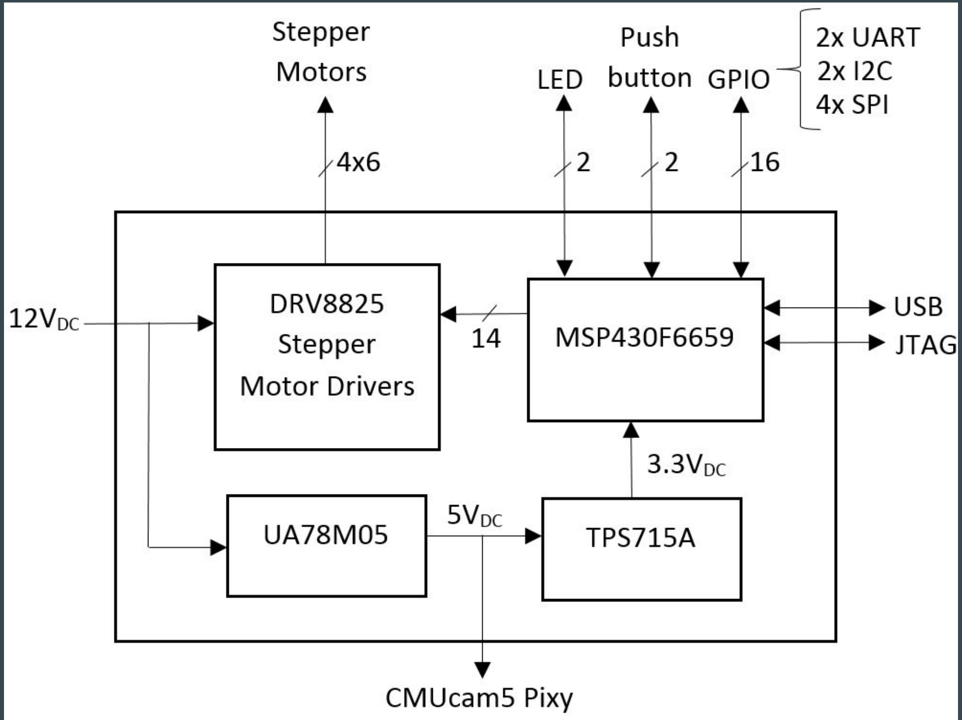


Electrical Design - Processor

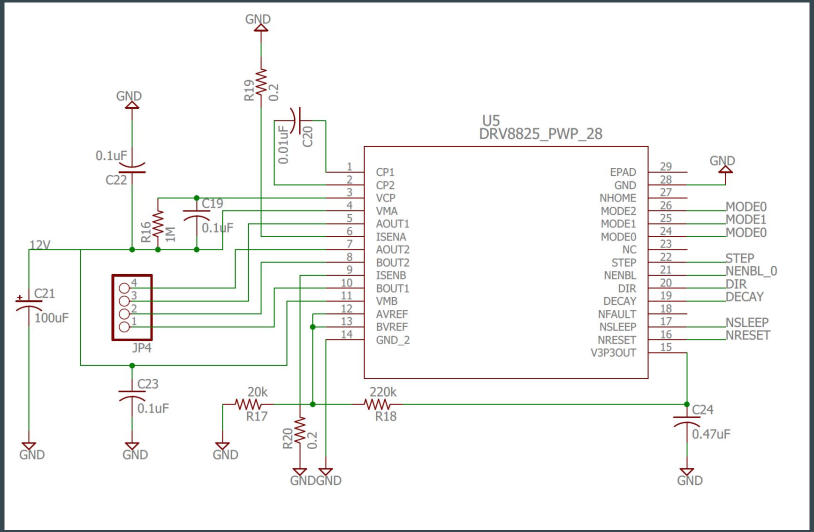
	MSP430G2553	MSP430F5529	MSP430F6659
ROM	16kB	128kB	512kB
RAM	500B	10kB	66kB
Serial	1 I2C, 1 UART	2 I2C, 2 UART	3 I2C, 6 UART
Extras	Temp Sensor	LCD & USB support	LCD & USB support
Power	230 µA/MHz	370 µA/MHz	404 µA/MHz
Price	$3	$8	$12

Electrical Design - Embedded System

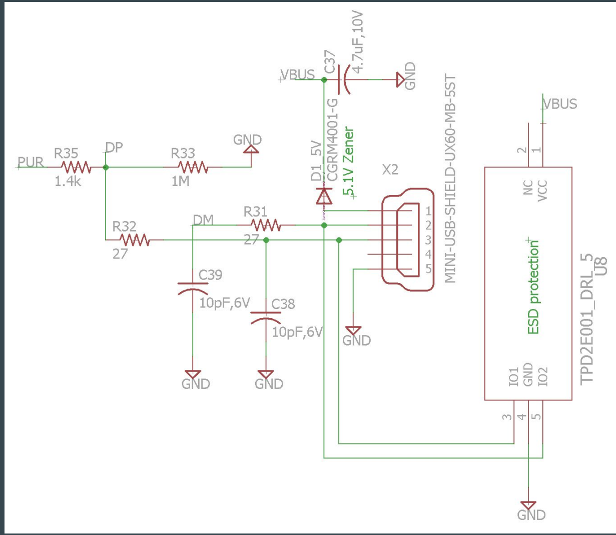
- MSP430
- 6 Stepper Driver ICs
- Mini USB
- 12V DC input
- 16 GPIO (can be internally mapped for serial communication)
- JTAG, SBW
- 2 user switches & LEDs
- 12V, 5V and 3.3V Headers



Electrical Design - Embedded System

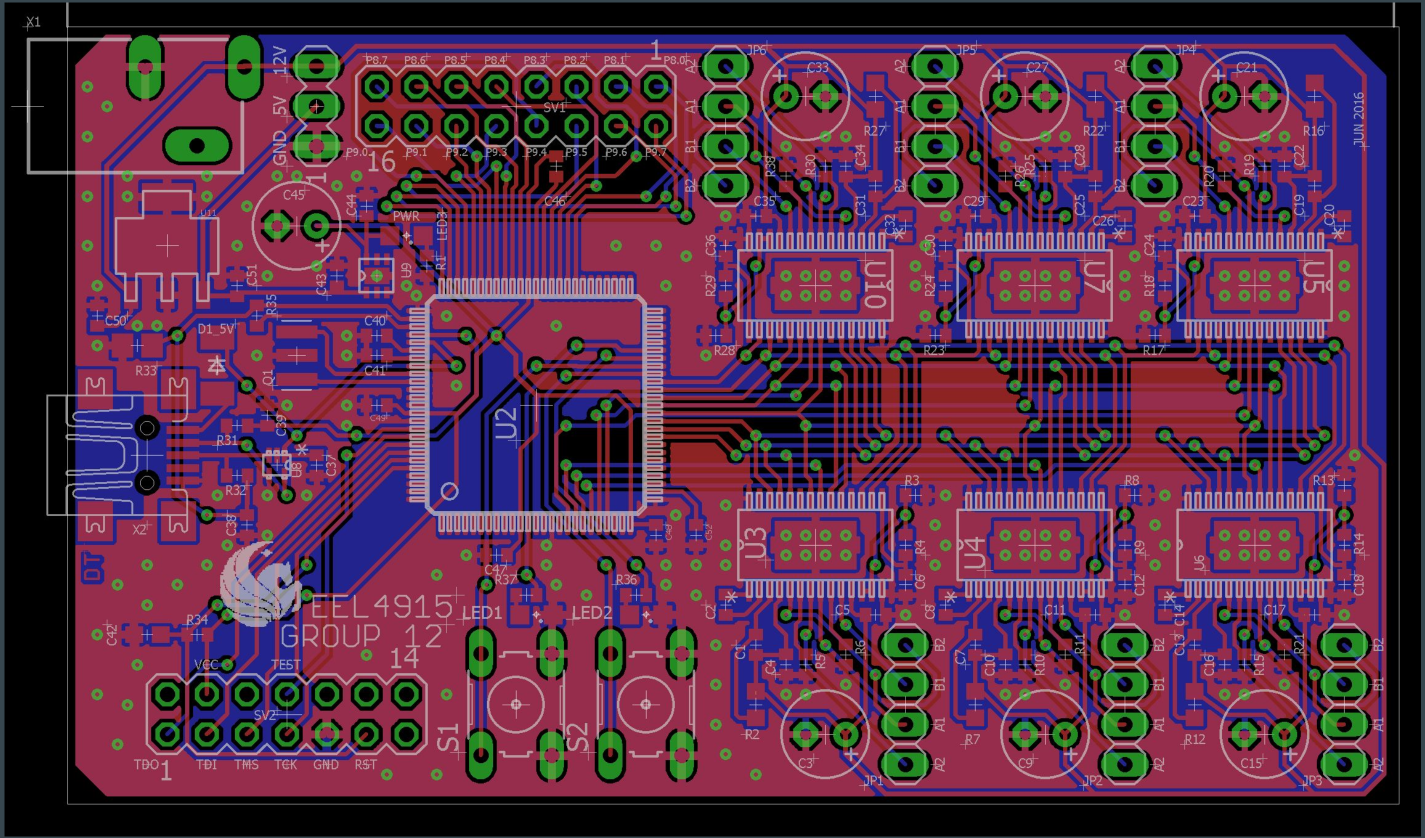


Stepper Motor Driver



USB Mini

Electrical Design - PCB



Electrical Design - PCB

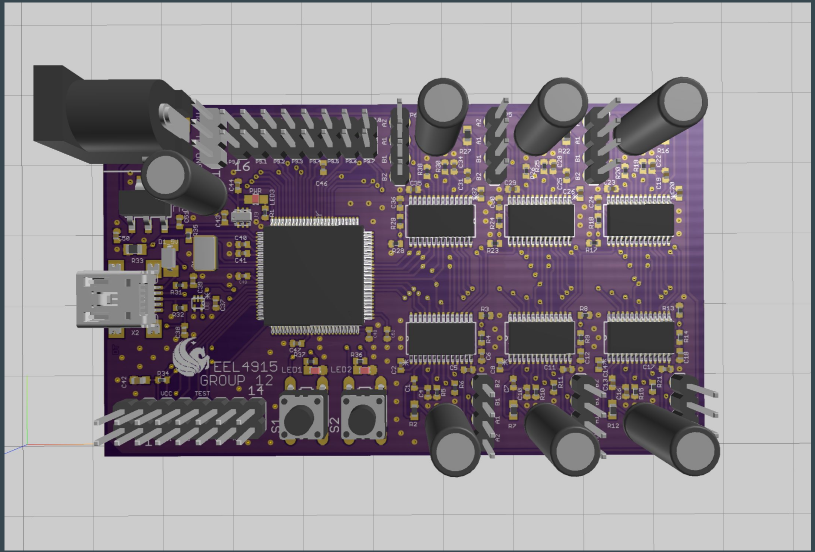


Image Sensing with Pixy Cam

Pixy is positioned to capture 6 tile faces along a single edge of the cube

Bright LEDs ensure that lighting is constant and stable during device operation

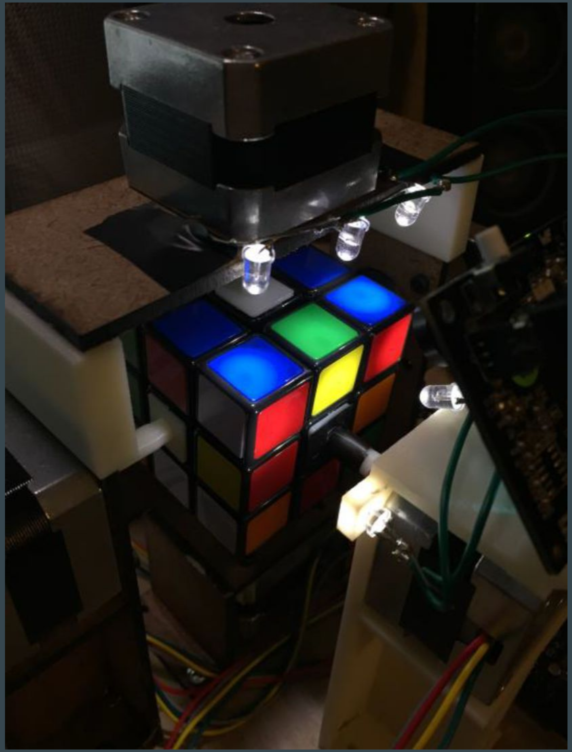


Image Sensing with Pixy Cam

View From Pixy Cam

Detects 6 **color signatures** for each color of the cube

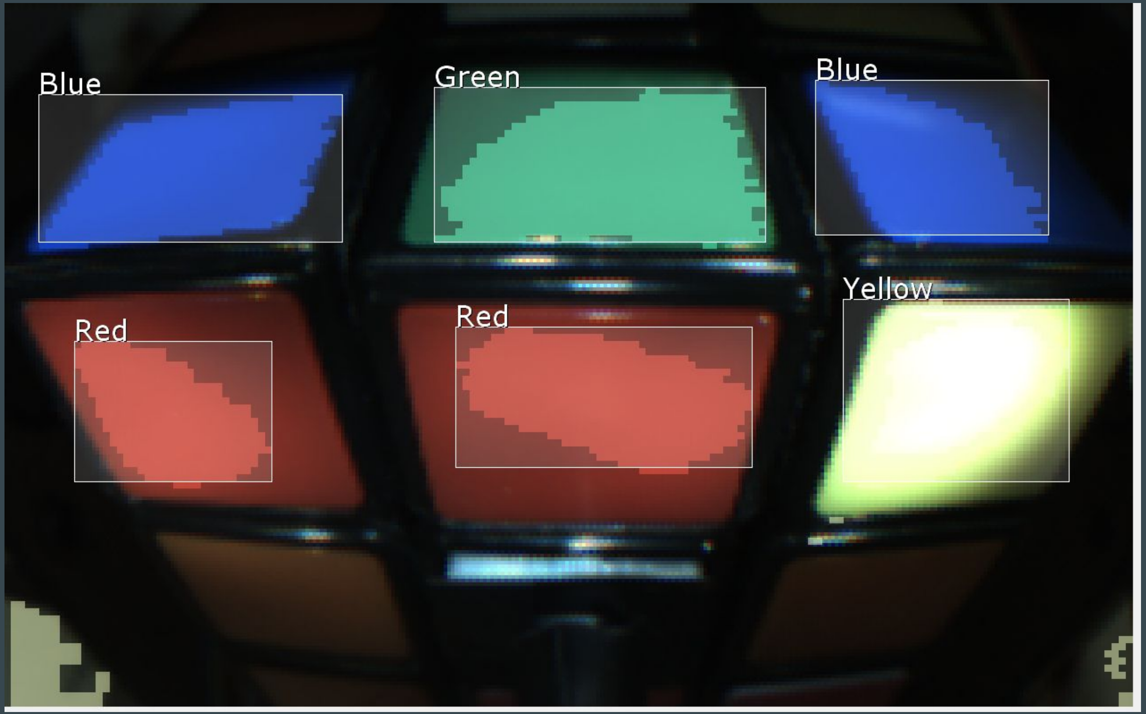


Image Sensing with Pixy Cam

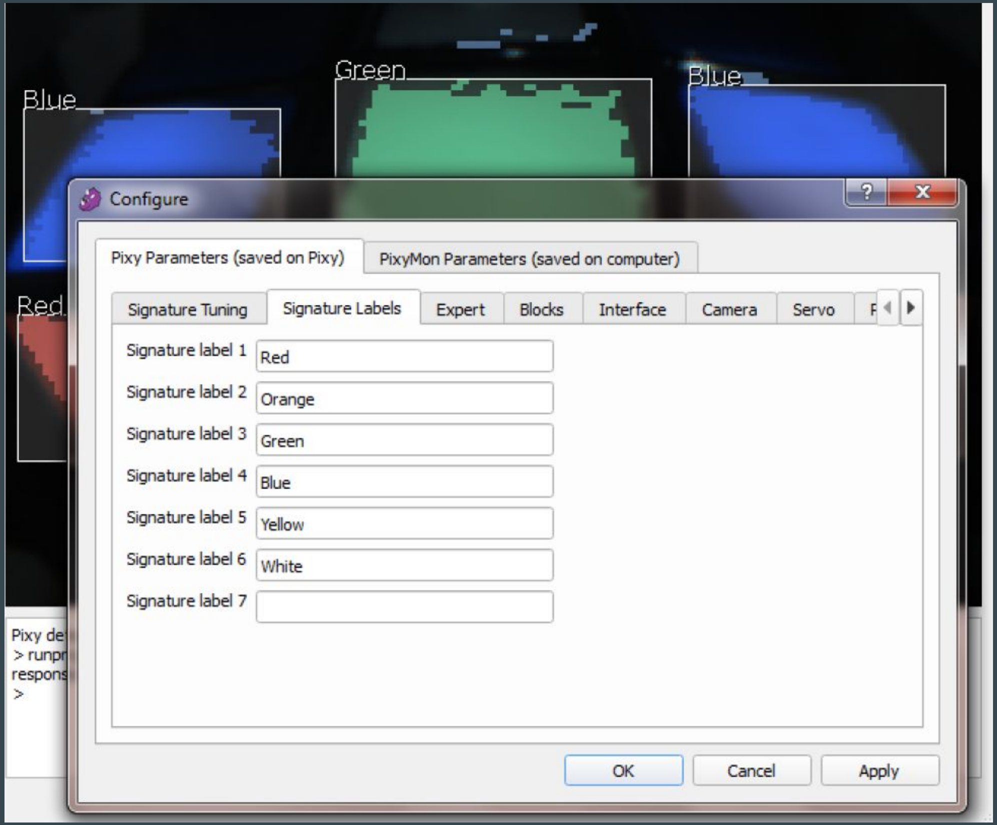


Image Sensing with Pixy Cam

- Detected colors are sent to MSP430 over UART connection
- Pixy use 50 frames per second:
- 50 * 6 * 14 * 8 ⇒ 33600 baud or greater
- ⇒ use 57600 baud

Bytes	16-bit word	Description
0, 1	y	sync: 0xaa55=normal object, 0xaa56=color code object
2, 3	y	checksum (sum of all 16-bit words 2-6)
4, 5	y	signature number
6, 7	y	x center of object
8, 9	y	y center of object
10, 11	y	width of object
12, 13	y	height of object

Image Sensing with Pixy Cam

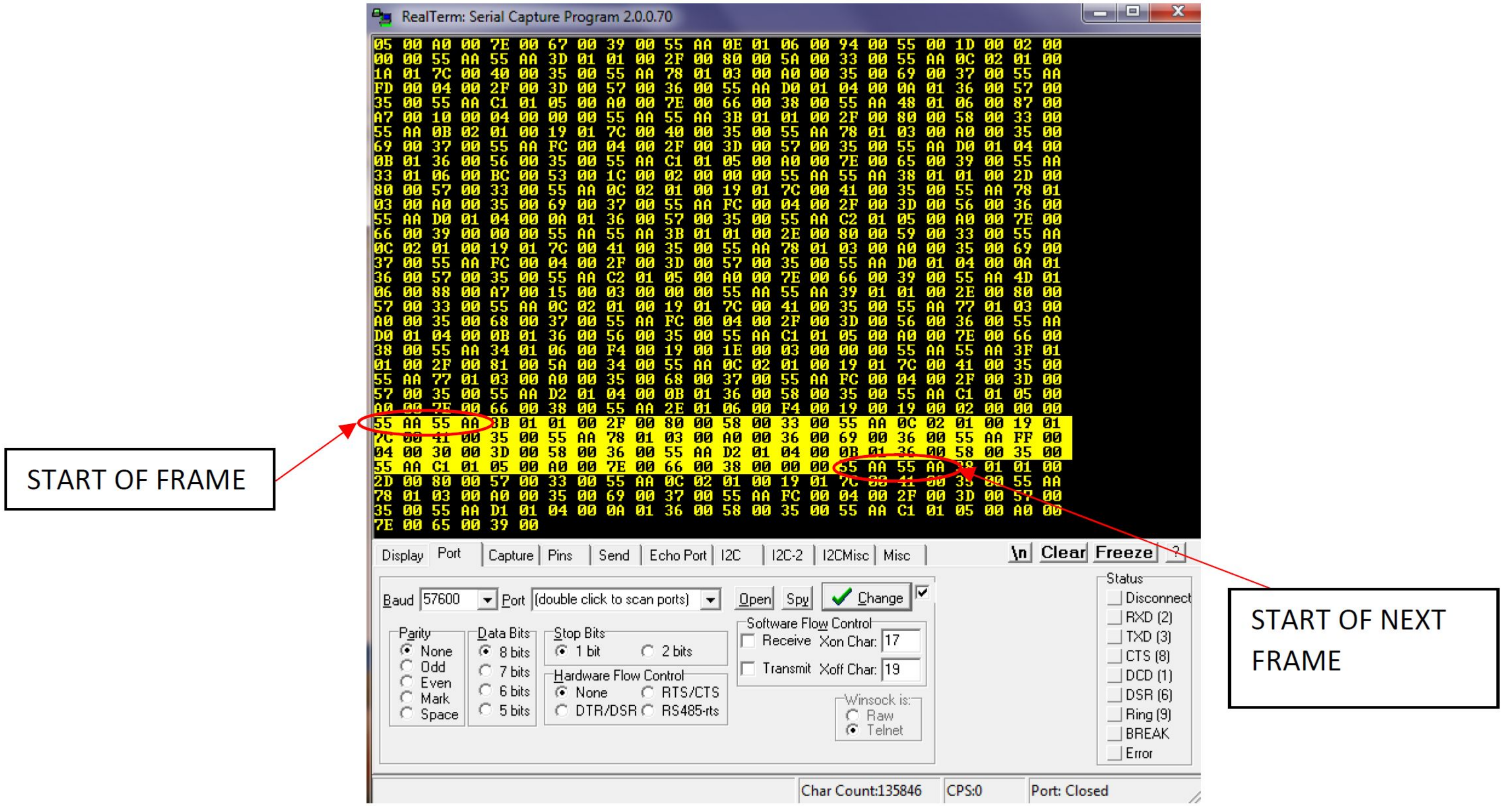
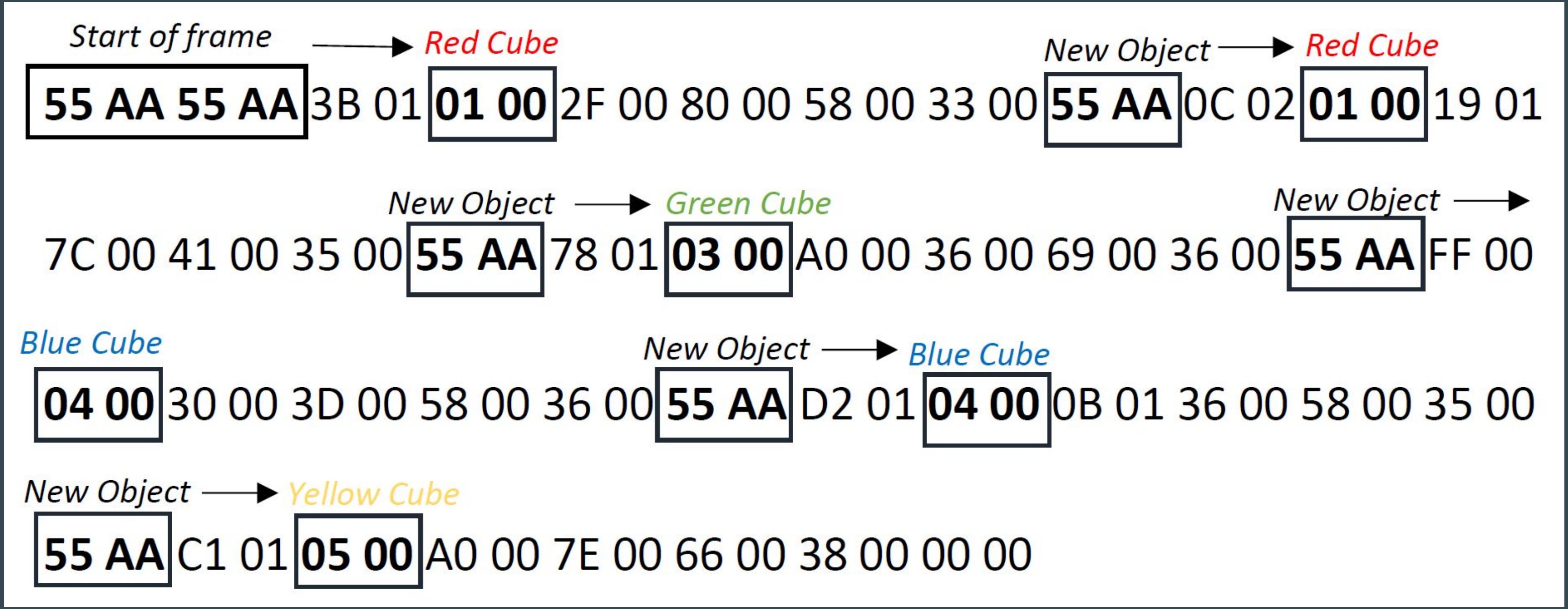


Image Sensing with Pixy Cam



Mapping the Cube

The orientation of each face of the cube.

We use the bytes found from the Image sensing to determine the orientation of the cube.

The bytes taken in from the Pixy CMU5 cam are the 4th byte, for the signature color, 6th and 7th, for the X position, and the 8th and 9th, for the Y position.

We determine from the position of our camera that we should receive 6 blocks per frame.

```
//UCF Senior Design Group 12
//int colors[5][8];
          |*********|
          |40 41 42 |
          |*********|
          |43 44 45 |
          |*********|
          |46 47 48 |
          |*********|
|*********|*********|*********|*********|
|00 01 02 |10 11 12 |20 21 22 |30 31 32 |
|*********|*********|*********|*********|
|03 04 05 |13 14 15 |23 24 25 |33 34 35 |
|*********|*********|*********|*********|
|06 07 08 |16 17 18 |26 27 28 |36 37 38 |
|*********|*********|*********|*********|
          |*********|
          |50 51 52 |
          |*********|
          |53 54 55 |
          |*********|
          |56 57 58 |
          |*********|
```

Mapping the Cube cont.

After we receive all 30 bytes of code that make up a frame we parse through it to determine the positions of each significant color in the picture.

We start by putting all the signature colors in an array of 0 to 5 corresponding with its X position and Y position.

We then convert our X positions and Y positions from hex to decimal to help us with their position in the frame.

The top right cornor of the frame is considered to be (0,0) in the X, Y plane.

As you travel along the axis of the frame, X and Y becomes respectively larger the further away you get.

We use this information to determine where the blocks in the frame lay.

Mapping the Cube cont.

For the X coordinate given by the Pixy CMU5cam:

if(X[i] <100)

Left most frame

else if(X[i] > 200)

Right most frame

else

Middle frame

For the Y coordinate given by the Pixy CMU5cam:

if(Y[i] < 100)

Top half of cube

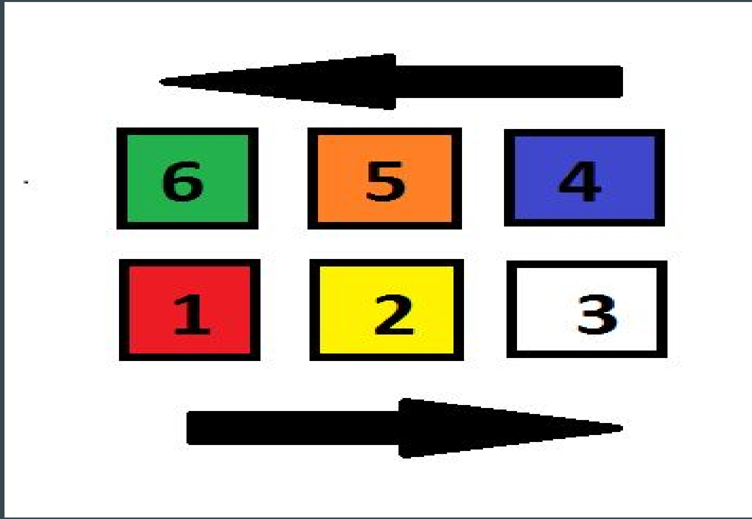
else

Bottom half

Mapping the Cube cont.

Using the pseudo code on the previous slide we can determine the positions of the 6 different blocks in each frame.

We then place the signature color in order from the bottom left-most block across to the bottom right-most block, and then return from top right-most to the top left-most.



The frame above would read in as

(R, Y, W, B, O, G)

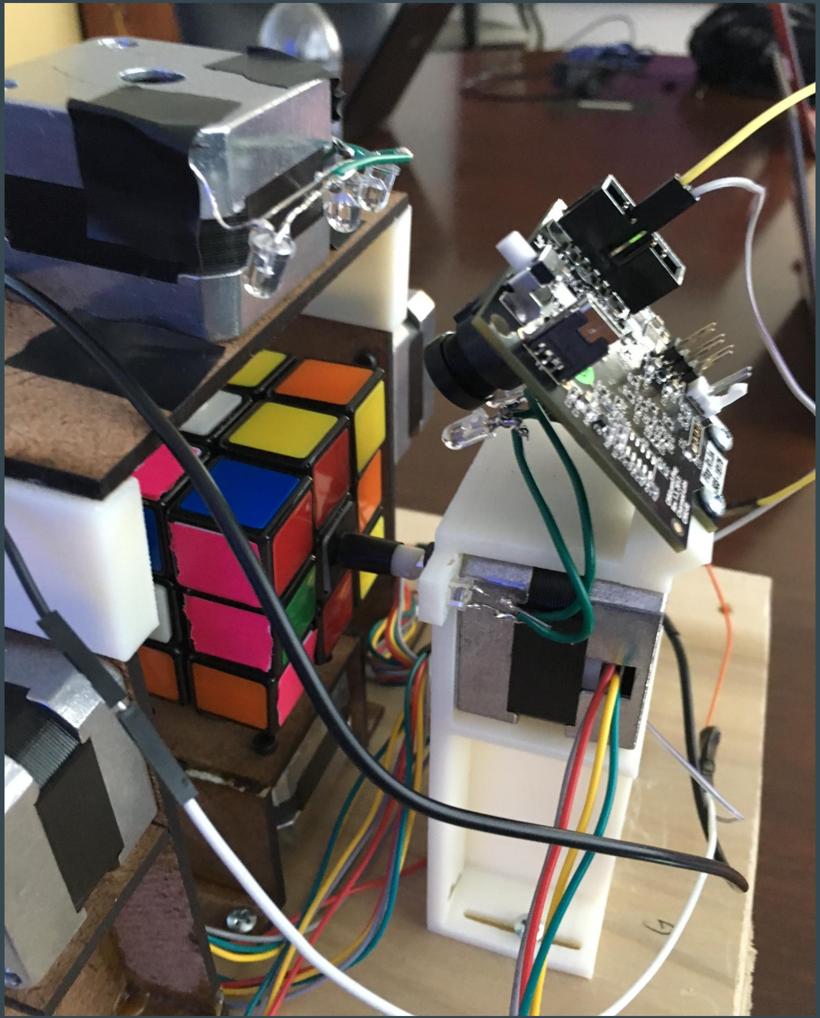
Mapping the Cube cont

Using this technique we can determine the orientation of the entire cube from start to finish by doing a specific set of movements.

After these movements we can then reset the cube back to its original formation before the mapping began allowing us to manipulate to solve for the correct orientation.

There is 12 different frames needed to take into visualize the entire cube from the position seen.

After each frame a specific set of moves is made to get to the next frame.



Serial OutPut/Input

For our GUI to communicate with our robot we had to establish a serial port connection and talk to it to receive and send data.

Input:

We need to use the serial port input to receive the data of each frame for our vision control.

We initialize the input command by sending a ‘!’ to our msp430 to tell it we need to start receiving camera information

Output:

We use the serial port output to send to our msp430 which function we are doing and what rotations need to be done by the robot.

send(‘!’)

Start camera input

send(‘2’)

Start rotation parsing

send(“FFRRDDLLUUBB”)

Sends our rotations

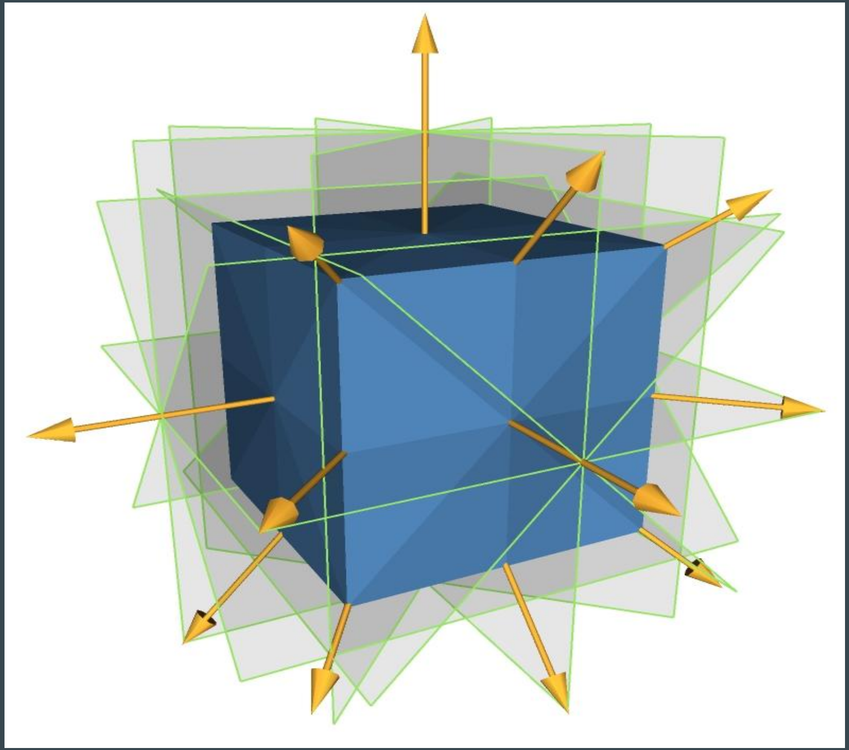
Math - Symmetries

Over 43 quintillion possibilities of the Rubik's Cube!

8 corners with 8! ways they can be arranged and seven corners that can be arranged independently with the eighth being dependent on the preceding seven

12 edges with 12!/2 ways they can be arranged. Divided by two because of its dependency to be even exactly when the corners are. And eleven of the edges can be moved independently. These together make the equation below:

8! x 3^7 x 12!/2 x 2^11 = 43,252,003,274,489,856,000



Math - Conjugations

Conjugate - binomial form by negating the second term of the binomial (conjugate of x+y is x-y)

Many of the algorithms of Rubik's Cube are derived from conjugates

Using David Singmaster Notation for the faces (Front - F, Left - L, Right - R, Up - U, Down - D, Back - B) and add prime symbol ' to a letter to denote CCW move

Example: Attempting to only change the U face of a solved cube R U R' U' will change 2 cubes that are not on the U face as displayed in Figure 1

Therefor a move F is added before it to orient those two cubes first then F' added to the end making only the U layer changed as displayed in Figure 2

Figure 1

Figure 2

Algorithm Options

CFOP (Speed Cubing)

- Minimum memory requirements
- Simple to develop
- Less efficient with time
- Solves the cube in hundreds of moves

Kociemba (God's Algorithm)

- Abuses RAM
- Complex development (need bluetooth and mobile app)
- More efficient with time
- Solves the cube in at most 20 moves

PC vs MSP430

Gave us GUI options to solve the cube

Not completely reliant on the vision working perfectly

Allowed for manual input of the cube

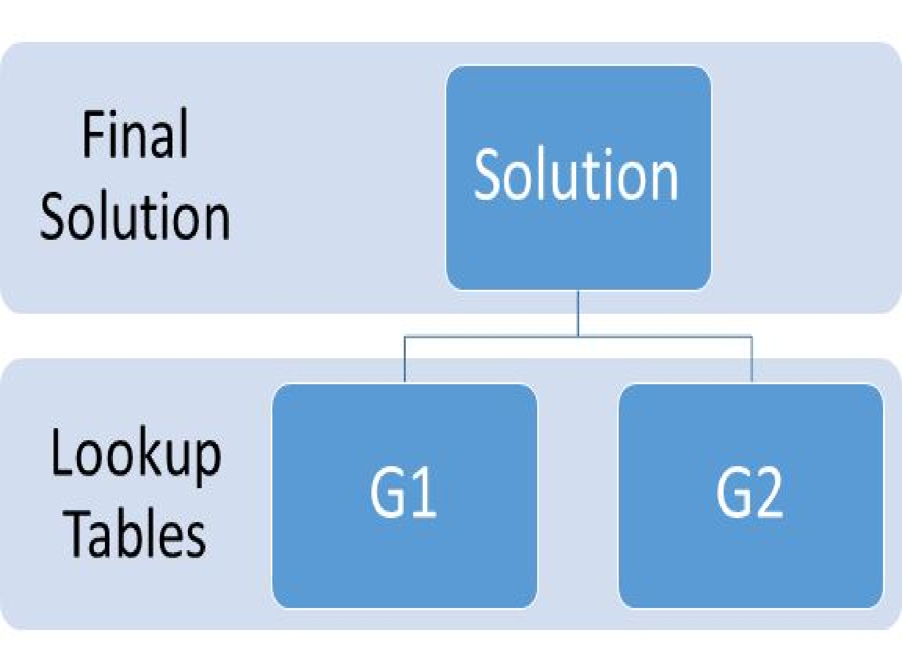
Gave us more power for a more efficient algorithm

Kociemba’s Algorithm

G1 is the first group and stores look up tables (millions of them) to find a solution for its state

G2 is the second group that stores look up tables to find a solution for its state

Both continually look for solutions that don’t include each other



Kociemba Groups

G0 state is simply the initial state of the cube

G1 state looks for many symmetries

This speeds up the Iterative Deepening because you can search multiple things at once

G2 only uses a specific moveset to iterate through the rest of the cube

Iterative Deepening

Is the primary engine behind the algorithm

It tries all solution that takes 15 moves for G1 and 9 moves for G2

It tries all solutions that take 16 moves for G1 and solves 8 moves for G2

...

Until it tries all solutions that take 24 moves and solves 0 moves for G2

Pruning

Is the main way to handle the speed

The millions of tables saved takes a lot of time and memory

Turning a face only has three possible states CCW, Idle, CW

Only store cubes moves by mod 3 to account for there only being 3 statess

Solving String

After the cube is solved for the correct moves we save make a string of every rotation needed to solve the cube.

The string consist of the order of rotations each considered to be clockwise unless they are followed by an ‘ which then would result in a counterclockwise rotation

Scramble

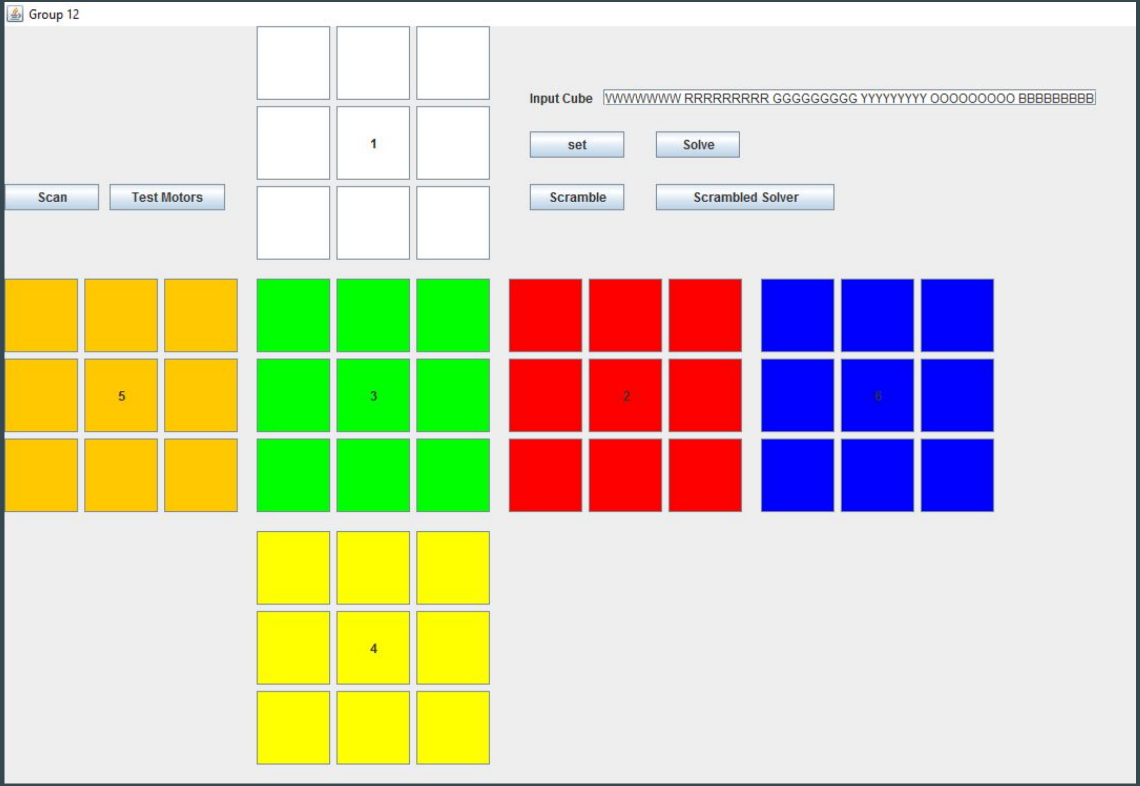
- Can’t take out the cube when it’s in our robot
- Need to scramble the cube after its solved
- Easier to just make a program that randomizes the cube
- We plan on using a random number generator to randomize the cube

GUI/Display

2D display of the cube

Buttons for actions of the cube

Text box to input the cube



Budget

5		Amazon 5/18	1 Stepper motor and 5 stepper driver chips for prototype	$45.91				Pixy Cam		$70.00
6		Digikey 5/24	5 Stepper motors and a few small misc connectors/header pins	$103.41						
7		Amazon 5/25	Rubik's Cube and remaining stepper driver chips for prototype	$38.12						
8		Digikey 6/9	Main PCB component purchase	$142.88						
9		Oshpark PCB	PCB Fabrication	$37.50						
10		Home Depot	Wood, screws, glue for building structure	$17.07						
11	Individual Total			$384.89			$0.00			$70.00
12	Total Cost:	$454.89								

Group Distribution

	Structure	Vision Control	Algorithm	GUI	Electronics
Daniel					
Corey					
Tony					

Primary Job **Secondary Job**

Trials and tribulations

- Pixy CMU5cam
  - Require high degree customization.
  - Better for object detection(motion tracking) than color orientation
  - Has issues with differentiating colors of close hues. (Red, Orange)(Yellow, White)
  - Documentation is out of date
  - Pre-built libraries for Arduino and lego not much so for customization

Conclusion

- Robot consistently and accurately solves the cube in under 10 seconds
- Cube Visualization is promising and yields preliminary results
- Versatile microcontroller-oriented PCB design

Questions?